

Corna Giorgio 1074241

Rossi Diego 1073945

# *FOOD APP*

**Progetto di  
ingegneria  
del software**



# Obiettivo

- Il progetto riguarda lo sviluppo di un'applicazione rivolta a ristoranti e bar.
- In particolare, l'app fornisce la gestione delle comande prese dai camerieri, sia che lavorino appunto in un ristorante oppure in un bar.
- Inoltre, l'app fornisce un'interfaccia anche per la cucina, la quale può vedere gli ordini presi.

# Difficoltà incontrate

Le difficoltà incontrate sono state principalmente tre:

- Implementazione del database
- Compatibilità di Android Studio tra Windows e Mac
- Test dei metodi del Database.

# Paradigma di programmazione/modellazione utilizzato e tools

- Ambiente di sviluppo —————> Android Studio
- Linguaggio di programmazione —————> Java
- Modellazione —————> StarUML
- Database —————> SSMS/Linguaggio SQL

# Software configuration management

- Tool utilizzato: Github

E' servito per organizzare il progetto, condividere il codice e documentazione tramite le pull requests e porre/risolvere problem tramite gli issues.

# Software life cycle

- Il software life cycle che abbiamo utilizzato è stato eXtreme Programming
- Il modello che abbiamo utilizzato per organizzare le fasi da svolgere è stato il modello a cascata.
- Durante la fase di implementazione, abbiamo seguito un modello Moscow per decidere quali funzioni implementare prima.



# Qualità

- La gestione della qualità è stata gestita seguendo alcune qualità definite nel modello ISO 9126, quali:
- 1)MUTEVOLEZZA
- 2)COMPORTAMENTO TEMPORALE
- 3)OPERABILITA'

# Requisiti

- I requisiti sono stati distinti in funzionali e non funzionali.
- Entrambi suddivisi in:
  - Must have
  - Should have
  - Could have
  - Won't have



### 3.1) REQUISITI FUNZIONALI

1. MUST HAVE: NUOVA PRENOTAZIONE, VISUALIZZA PRENOTAZIONE, NUOVO ORDINE, VISUALIZZA ORDINE
2. SHOULD HAVE: MODIFICA PRENOTAZIONE, MODIFICA ORDINE, CANCELLA PRENOTAZIONE, CANCELLA ORDINE
3. COULD HAVE: CHECK PRENOTAZIONE, CHECK ORDINE
4. WON'T HAVE: LISTA PIATTI CON ALLERGENI, LISTA TAVOLI, NOTE ALL'INTERNO DELL' ORDINE

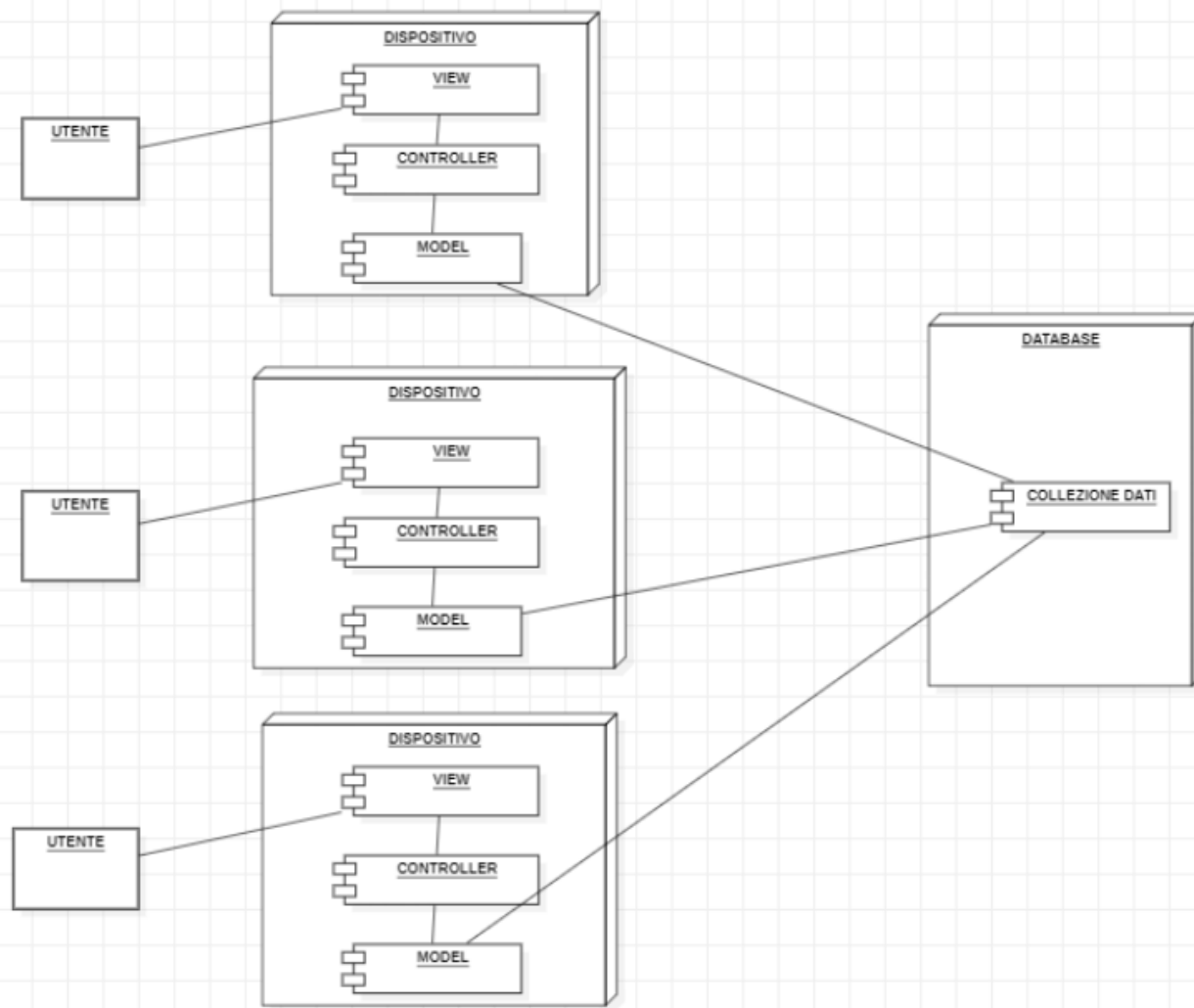
### 3.2) REQUISITI NON FUNZIONALI

1. MUST HAVE: interfaccia semplice ed intuitiva per non creare difficoltà nell' utilizzo
2. SHOULD HAVE: integrità dei dati nel database per non confondere ordini e prenotazioni.
3. COULD HAVE: tempi di risposta brevi (nell' ordine dei secondi).
4. WON'T HAVE: gestione della concorrenza nell' accesso al database.

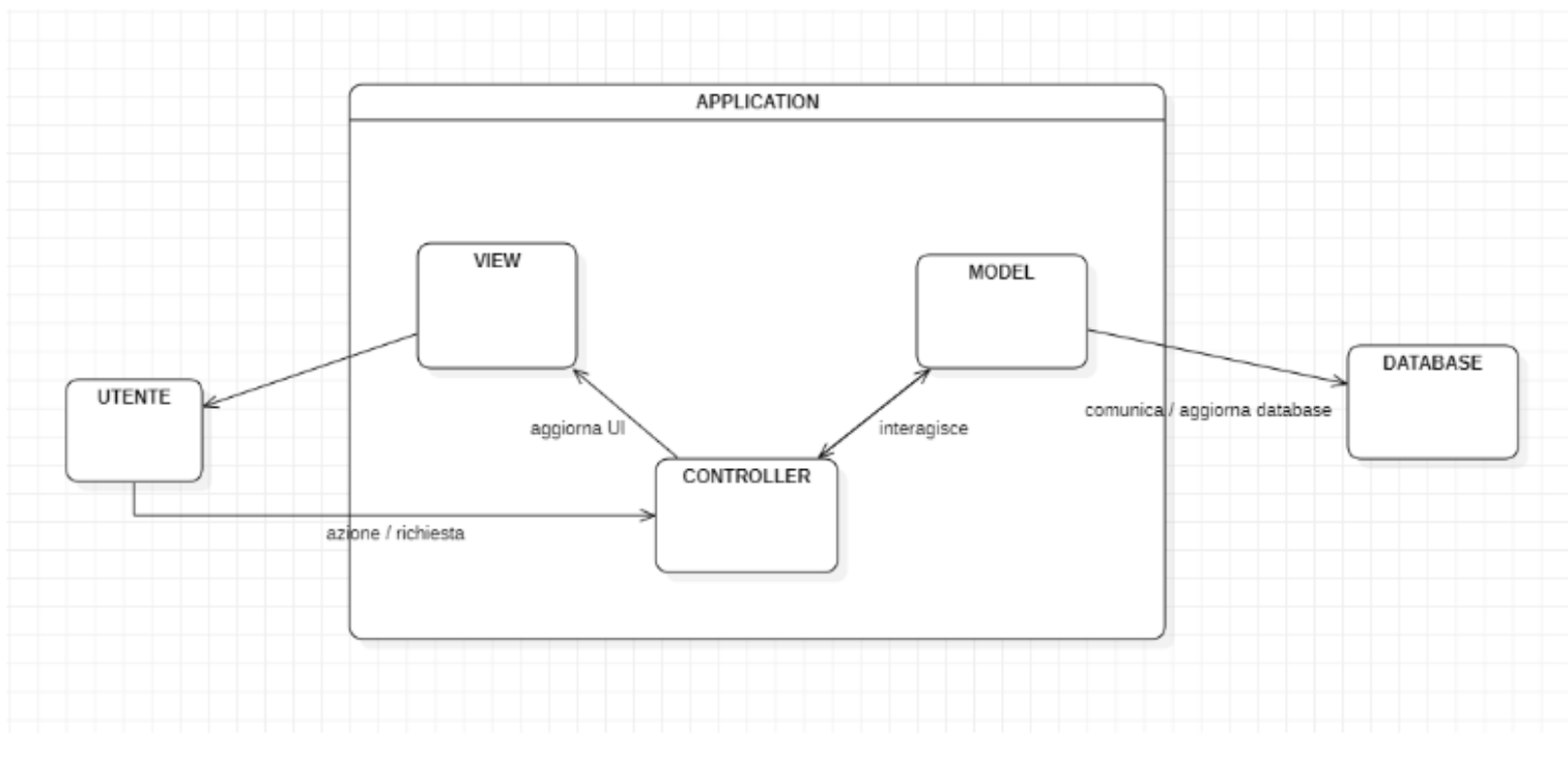
# Architettura

- Lo stile architetturale che abbiamo deciso di seguire è stato il model view controller.
- Esso permette di mantenere un codice ordinato, facile da leggere e facilmente modificabile in futuro.

## VISTA STATICA

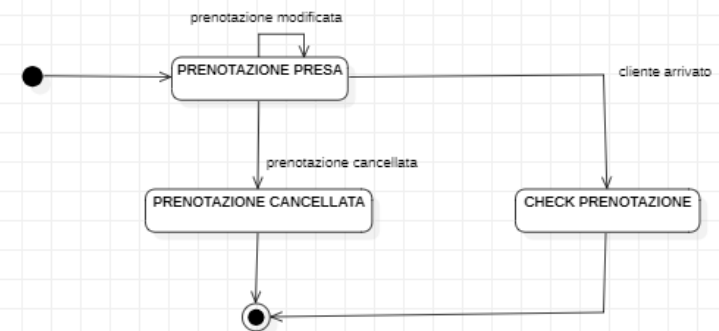
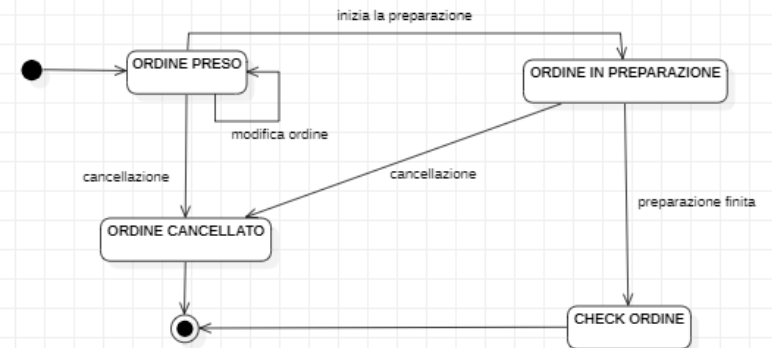
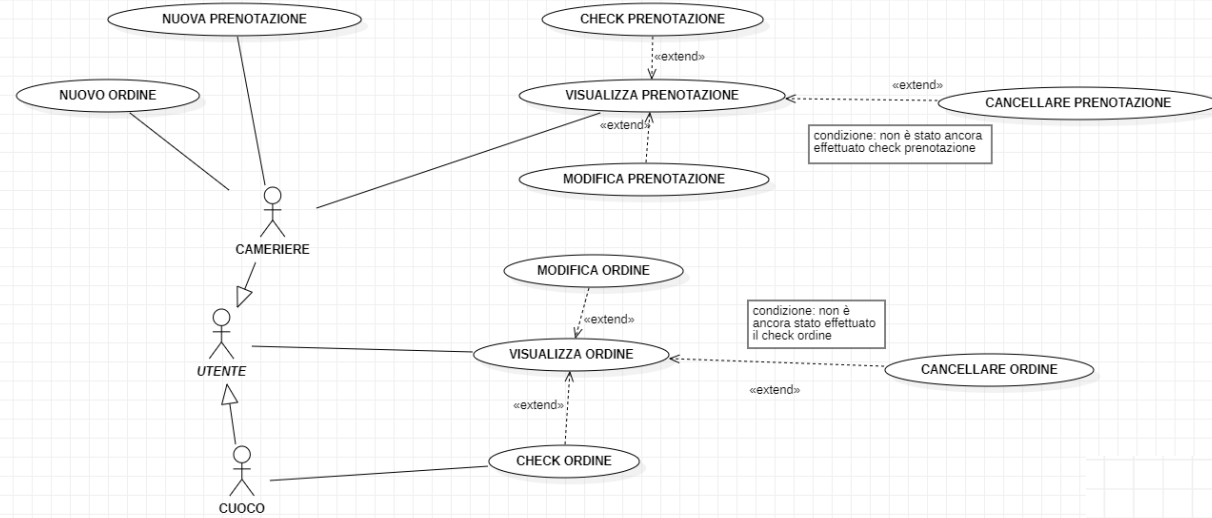


## VISTA DINAMICA

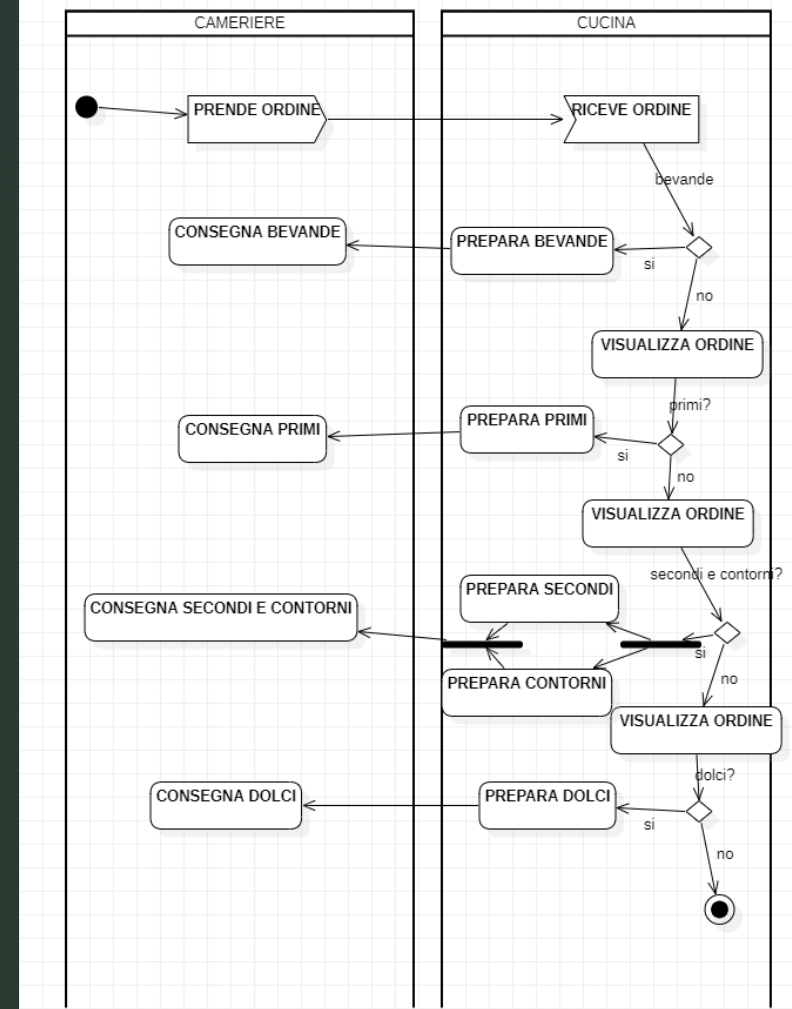
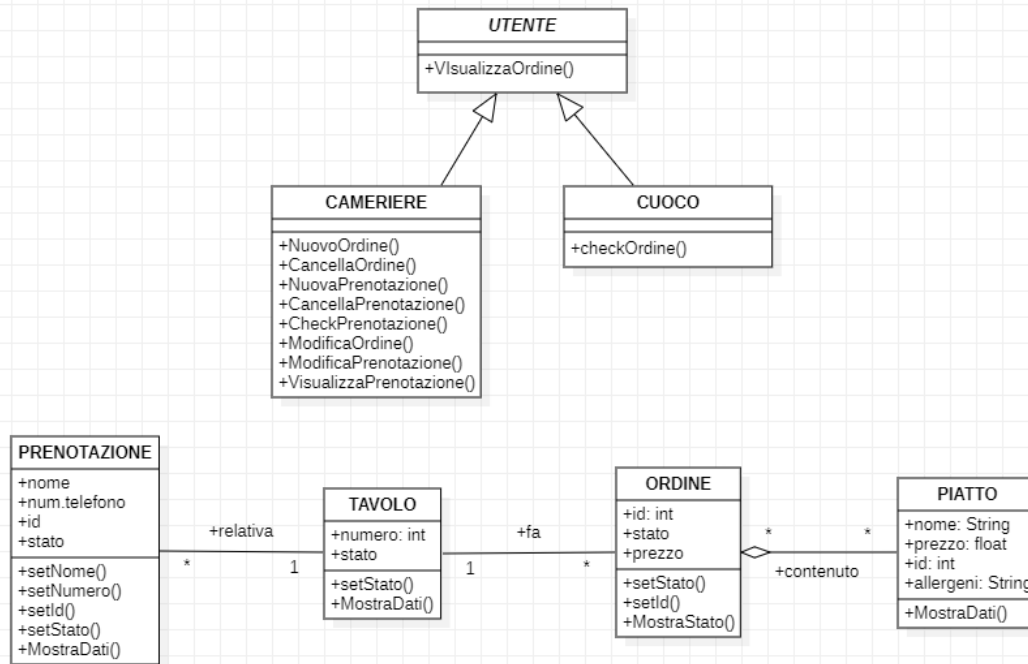


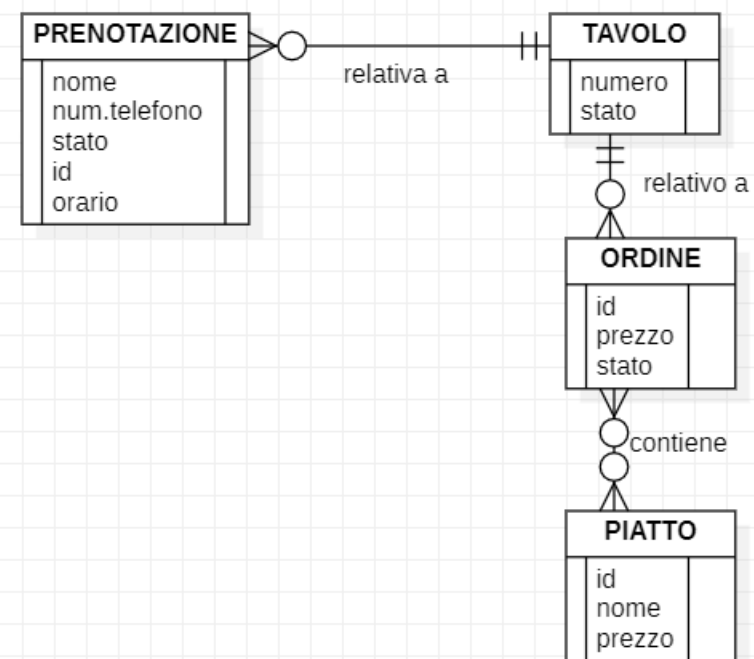
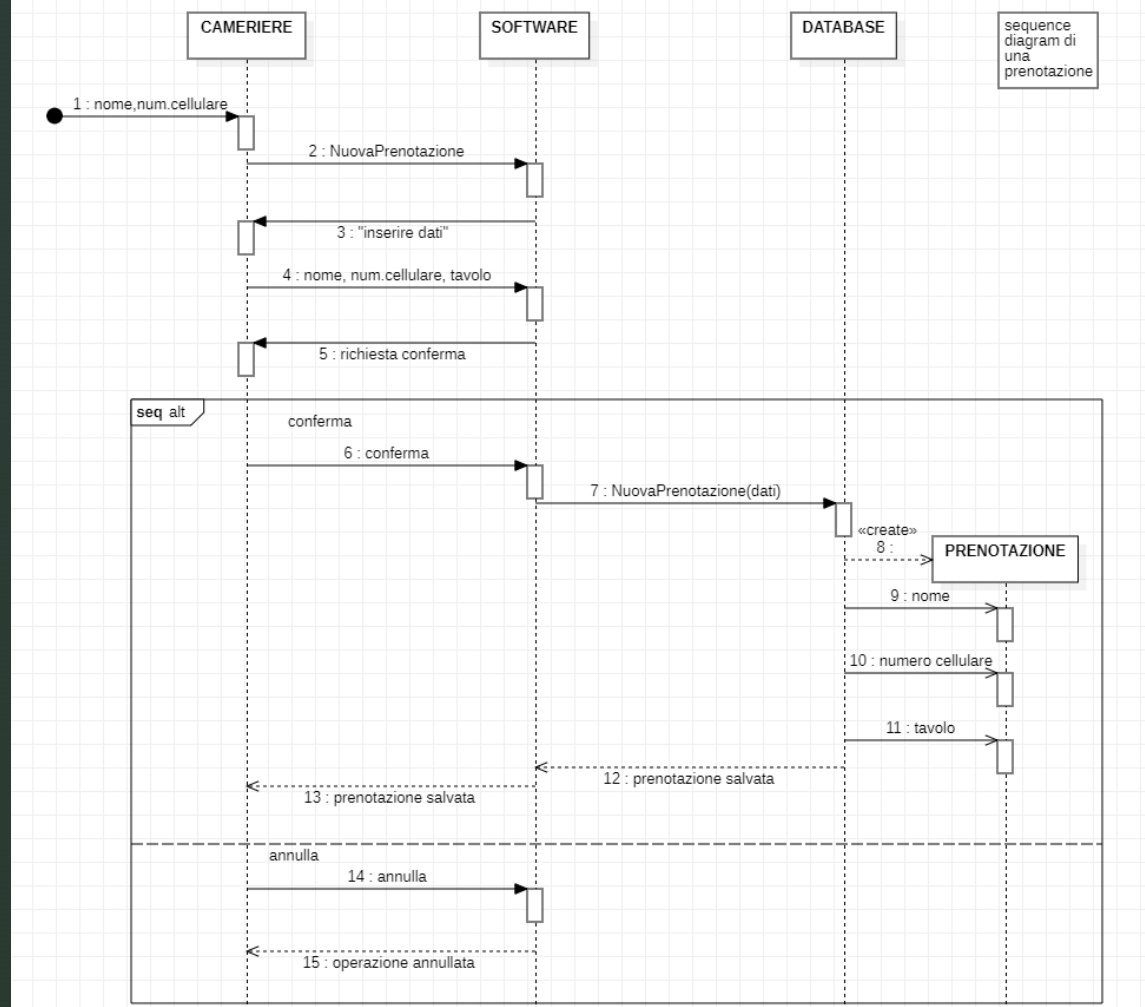
# Modellazione

- Tramite starUML abbiamo modellato i seguenti diagrammi:
  - Casi d'uso
  - Attività
  - Classi
  - Sequenza
  - Stati



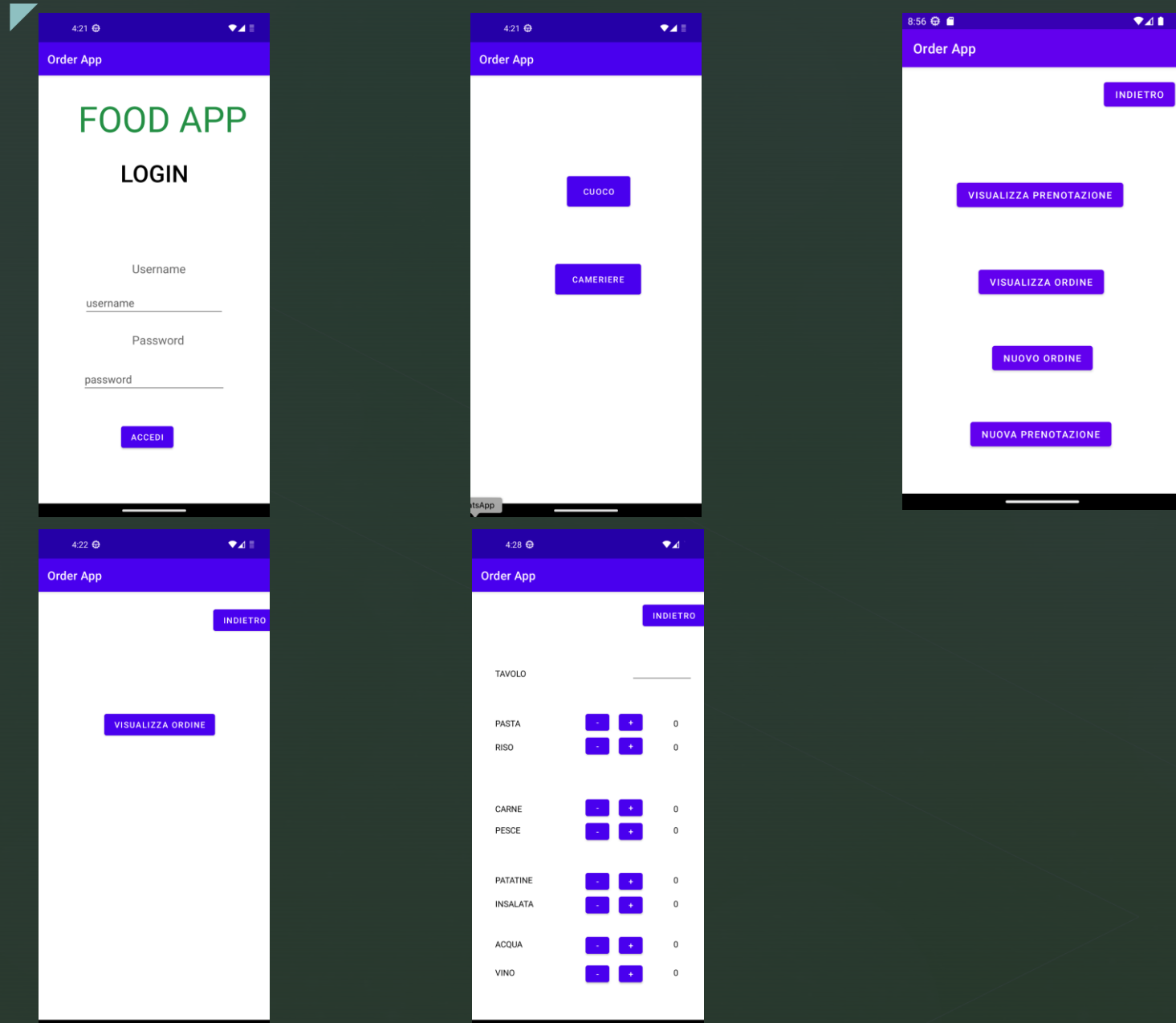






# Implementazione

- Sono stati implementati principalmente i requisiti funzionali **MUST HAVE** descritti nel piano di progetto e nella documentazione dei requisiti.
- Ciò che non è stato implementato del tutto sono i vari requisiti **SHOULD/COULD/WONT' HAVE**



DEMO

## DEMO

7:52

Order App

LISTA ORDINI

ID	PREZZO	STATO	TAVOLO
49	12.0	null	5
50	22.0	null	15
51	22.0	null	5
52	33.0	null	78
53	34.0	null	8
54	23.0	null	7

8:07

Order App

ORARIO:

NOME:

CELLULARE:

TAVOLO:

AGGIUNGI

7:59

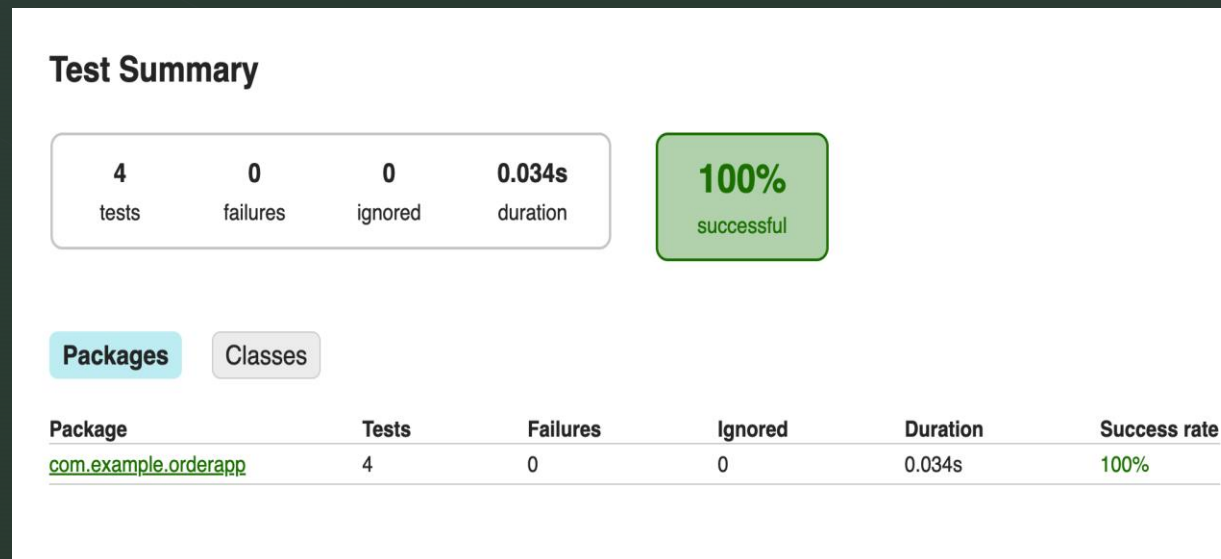
Order App

ELENCO PRENOTAZIONI

ID	ORARIO	NOME	CELLULARE	STATO	TAVOLO
29	21:00	DIEGO	3313334490	null	12
30	19:15	GIOVANNI	3314587896	null	3

# Testing Login

Per la fase di testing abbiamo utilizzato Junit.



Test per il Login



# Testing Database

## Test Summary

<b>3</b> tests	<b>3</b> failures	<b>0</b> ignored	<b>0.010s</b> duration	<b>0%</b> successful
-------------------	----------------------	---------------------	---------------------------	-------------------------

**Failed tests**

Packages

Classes

DbTest. CalcoloPrezzoTest

DbTest. InserimentoOridneTest

DbTest. InserimentoPrenotazioneTest

# Manutenzione del software

- Dopo aver implementato ogni funzione abbiamo svolto una pratica di Refactoring per rendere il codice più leggibile e più concorde alle qualità seguite.